1. **What is ReactJS?**

   ReactJS is open source JavaScript library used for building reusable UI components. We will use it to build the view component. Everything in the React is a component.

2. **Explain the history of ReactJS?**

   It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

3. **Why ReactJS?**

   The previous frameworks (Angular, Node) follow the traditional data flow structure, which uses the DOM (Document Object Model). DOM is an object which is created by the browser each time a web page is loaded. It dynamically adds or removes the data at the back end and when any modifications were done, then each time a new DOM is created for the same page. This repeated creation of DOM makes unnecessary memory wastage and reduces the performance of the application.

   ReactJS framework invented which remove this drawback. ReactJS allows you to divide your entire application into various components. ReactJS still used the same traditional data flow, but it is not directly operating on the browser's Document Object Model (DOM) immediately; instead, it operates on a virtual DOM. After the virtual DOM has been updated, React determines what changes made to the actual browser's DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM; instead, we are writing virtual components that react will turn into the DOM.

4. **Explain advantages and disadvantages of ReactJS?**

   Advantages:

   - ✓ Easy to Learn and Uses
   - ✓ Reusable components
   - ✓ Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
   - ✓ Scope for Testing the Codes

   Disadvantages:

   - ✓ Poor Documentation
   - ✓ Regular releases

- ✓ JSX as a barrier: ReactJS uses JSX. It's a syntax extension that allows HTML with JavaScript mixed together. This approach has its own benefits, but some members of the development community consider JSX as a barrier, especially for new developers. Developers complain about its complexity in the learning curve.

5. What was the latest version of ReactJS and its features?

Latest version is 17.0.2.

- ✓ Clean up in useEffect() will change from synchronous to asynchronous
- ✓ Router 6 support

6. What are all the prerequisites for React JS?

- ✓ NodeJS
- ✓ NPM
- ✓ Visual Studio

7. Explain different ways to create the React Application?

NPM and node versions should be like below,

1. Node version >= 8.10
2. NPM version >= 5.6

We can create the react app in following ways,

First Way:

Install React

We can install React using npm package manager by using the following command. There is no need to worry about the complexity of React installation. The create-react-app npm package manager will manage everything, which needed for React project.

```
C:\Users\javatpoint> npm install -g create-react-app
```

Create a new React project

Once the React installation is successful, we can create a new React project using create-react-app command. Here, I choose "reactproject" name for my project.

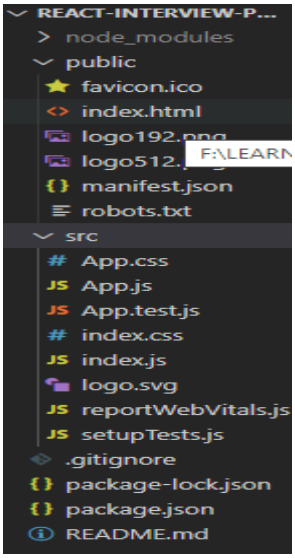
```
C:\Users\javatpoint> create-react-app reactproject
```

Second Way:

```
C:\Users\javatpoint> npx create-react-app reactproject
```

NOTE: We can combine the above two steps in a single command using **npx**. The npx is a package runner tool which comes with npm 5.2 and above version.

8. Explain about ReactJS project structure?



1. **node_modules:** It contains the React library and any other third party libraries needed.
2. **public:** It holds the public assets of the application. It contains the index.html where React will mount the application by default on the element.
3. **src:** It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files. Here, the App.js file always responsible for displaying the output screen in React.
4. **package-lock.json:** It is generated automatically for any operations where npm package modifies either the node_modules tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.
5. **package.json:** It holds various metadata required for the project. It gives information to npm, which allows to identify the project as well as handle the project?s dependencies.
6. **README.md:** It provides the documentation to read about React topics.

9. Explain the React features?

- ✓ **JSX:** JSX stands for JavaScript XML. It is a JavaScript syntax extension. It's an XML or HTML like syntax used by ReactJS. This syntax is processed into JavaScript calls of React Framework. It is not necessary to use JSX, but it is recommended to use in ReactJS.
- ✓ **Components:** ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls. These components can be reusable which help you to maintain the code when working on larger scale projects.
- ✓ **Virtual Dom:** Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then it checks the difference between the

previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

- ✓ **One-way Binding:** ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control throughout the application. Flux is a pattern that helps to keep your data unidirectional.
- ✓ **Simplicity:** ReactJS uses JSX file which makes the application simple and to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.
- ✓ **Performance:** ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM.

10. Explain more about JSX?

JSX provides you the way to write HTML/XML code along with JavaScript code. When we render the function preprocessor (**babel**) will transform these expressions into actual JavaScript code.

JSX File

```
<div>Hello JavaTpoint</div>
```

Corresponding Output

```
React.createElement("div", null, "Hello JavaTpoint");
```

The above line creates a **react element** and passing **three arguments** inside where the first is the name of the element which is div, second is the **attributes** passed in the div tag, and last is the **content** you pass which is the "Hello JavaTpoint."

11. Explain about React components?

In ReactJS, we have mainly two types of components. They are

- ✓ Functional Components
- ✓ Class Components

Functional Components: In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters.

Example:

```
function App() {
  return (
    <div className="App">
      <h1> Hello World</h1>
    </div>
  );
}

export default App;
```

Class Components: Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function.

```
import React from "react";

export default class ClassComp extends React.Component{
    constructor(props) {
        super(props)

    }

    render(){
        <div>Hello World</div>
    }

}
```

12. **Explain about React naming conventions?**

    HTML tags always use lowercase tag names, while React components start with Uppercase.

    Example: You should use className and htmlFor as XML attribute names instead of class and for.

13. **How to add comments in React?**

    When writing comments, we need to put curly brackets {} when we want to write comment within children section of a tag. It is a good practice to always use {} when writing comments.

```
class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Header</h1>
            {//End of the line Comment...}
            {/*Multi line comment...*/}
         </div>
      );
   }
}
export default App;
```

14. Explain about React state?

In React we will use the state to hold the information. We can update the data in state also. A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render.

A state represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly.

In general we will set the initial state in constructor of the component. We can update the state by using **setState**.

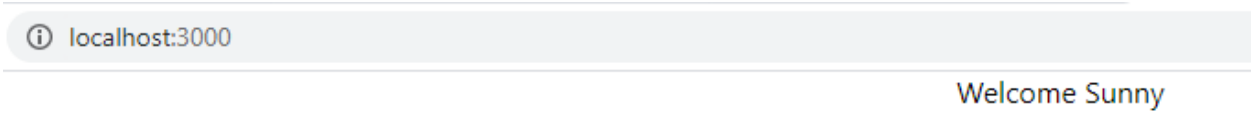
```
export default class ClassComp extends React.Component {
  constructor(props) {
    super(props);

    //defining the initial state
    this.state = { displayName: false };
  }

  //Updating the State
  showName() {
    this.setState({ displayName: true });
  }

  render() {
    {
      this.state.displayName ? "Test" : "";
    }
    <div>
      <button onClick={this.showName}>Show Name</button>
    </div>;
  }
}
```

15. Explain about the props?

Props stand for Properties. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function. **Props are read-only components.**

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as **this.props** and can be used to render dynamic data in our render method.

```
export default class PropsComponent1 extends Component {
  constructor(props) {
    super(props);

    this.state = {
      name: "Sunny",
    };
  }

  render() {
    return (
      <div>
        <ChildComponent nameProp={this.state.name} />
      </div>
    );
  }
}

class ChildComponent extends React.Component {
  render() {
    return <div>Welcome {this.props.nameProp}</div>;
  }
}
```



16. Why can't we update the React props?

First reason is React is one-way binding.

The React philosophy is that props should be immutable and top-down. This means that a parent can send whatever prop values it likes to a child, but the child cannot modify its own props. What you do is react to the incoming props and then, if you want to, modify your child's state based on incoming props.

So you don't ever update your own props, or a parent's props. Ever. You only ever update your own state, and react to prop values you are given by parent.

If you want to have an action occur on a child which modifies something on the state, then what you do is pass a callback to the child which it can execute upon the given action. This callback can then modify the parent's state, which in turns can then send different props to the child on re-render.

17. How to validate the props validation?

Props are an important mechanism for passing the read-only attributes to React components. The props are usually required to use correctly in the component.

Props validation is a tool that will help the developers to avoid future bugs and problems. It is a useful way to force the correct usage of your components. It makes your code more readable. React components used special property **PropTypes** that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes.

```
import PropTypes from 'prop-types';

export default class PropsValidations extends Component {
  render() {
    return (
      <div>
        <table>
          <tr>
            <th>Type</th>
            <th>Value</th>
            <th>Valid</th>
          </tr>
          <tr>
            <td>Array</td>
            <td>{this.props.propArray}</td>
            <td>{this.props.propArray ? "true" : "False"}</td>
          </tr>
          <tr>
            <td>Boolean</td>
            <td>{this.props.propBool ? "true" : "False"}</td>
            <td>{this.props.propBool ? "true" : "False"}</td>
          </tr>

          <tr>
            <td>String</td>
            <td>{this.props.propString}</td>
            <td>{this.props.propString ? "true" : "False"}</td>
          </tr>
          <tr>
            <td>Number</td>
            <td>{this.props.propNumber}</td>
            <td>{this.props.propNumber ? "true" : "False"}</td>
          </tr>
        </table>
      </div>
    );
  }
```

```
PropsValidations.propTypes = {
  propArray: PropTypes.array.isRequired,
  propBool: PropTypes.bool.isRequired,
  propNumber: PropTypes.number,
  propString: PropTypes.string,
};
PropsValidations.defaultProps = {
  propArray: [1, 2, 3, 4, 5],
  propBool: true,
  propNumber: 1,
  propString: "JavaTpoint",
};
```

Type	Value	Valid
Array	12345	true
Boolean	true	true
String	JavaTpoint	true
Number	1	true

18. **Difference between React state and props?**

State	Props
State is mutable.	Props are immutable.
State holds information about the components.	Props allow you to pass data from one component to other components as an argument.
State cannot be accessed by child components.	Props can be accessed by the child component.
States can be used for rendering dynamic changes with the component.	Props are used to communicate between components.
Stateless components cannot have State.	Stateless component can have Props.

19. **Explain about the constructor in React JS?**

A constructor is a special data member which can be invoked during the instantiation of a class component.
When you implement the constructor for a React component, you need to call **super(props)** method before any other statement. If you do not call super(props) method, this.props will be undefined in the constructor and can lead to bugs.

We will use the Constructor in ReactJS for below purpose,

- ✓ It used for initializing the local state of the component by assigning an object to this.state.
- ✓ It used for binding event handler methods that occur in your component.

Constructor implementation is mandatory neither initialize state nor bind methods.

You cannot call setState() method directly in the constructor(). If the component needs to use local state, you need directly to use 'this.state' to assign the initial state in the constructor. The constructor only uses this.state to assign initial state, and all other methods need to use set.state() method.

Constructor Example:

```
export default class ClassComp extends React.Component {

  //Added the Constructor
  constructor(props) {
    super(props);

    //defining the initial state
    this.state = { displayName: false };

    //Binding the events.
    this.showName = this.showName.bind(this)
  }

  //Updating the State
  showName() {
    this.setState({ displayName: true });
  }

  render() {
    {
      this.state.displayName ? "Test" : "";
    }
    <div>
      <button onClick={this.showName}>Show Name</button>
    </div>;
  }
}
```

20. In class component why we need to bind the events in constructor?

In ReactJS, we need to bind events so that this keyword would not return an “undefined“.

```
import React, { Component } from 'react';

class EventBind extends Component {
    constructor(props) {
        super(props)

        this.state = {
            message: 'Welcome'
        }
    }

    clickHandler() {
        this.setState({
            message:'Farewell'
        })
    }

    render() {
        return (
            <div>
                <h3>{this.state.message}</h3>
                <button onClick={this.clickHandler}>Click</button>
            </div>
        )
    }
}
```

Now if we run the application and click on the button, we get an error. This is because this returns an “undefined”. This is why we need to bind our events.

21. Explain about the arrow functions in ReactJS?

The Arrow function is the new feature of the ES6 standard. If you need to use arrow functions, it is not necessary to bind any event to 'this.' Here, the scope of 'this' is global and not limited to any calling function. So if you are using Arrow Function, there is no need to bind 'this' inside the constructor.

```
import { React, Component } from "react";

export default class ArrowFunctionsComponent extends Component {
  constructor() {
    super();
    this.state = {
      display: false,
    };
  }

  //Using arrow functions
  showName = () => {
    this.setState({ display: true });
  };

  render() {
    return (
      <div>
        {this.state.display ? "Welcome" : ""}
        <button onClick={this.showName}>Show</button>
      </div>
    );
  }
}
```

22. Is it mandatory to have constructor in class component?

No, it is not necessary to have a constructor in every component. Constructor implementation is mandatory neither initialize state nor bind methods.

23. Is it necessary to call super() inside a constructor?

Yes, it is necessary to call super() inside a constructor. If you need to set a property or access 'this' inside the constructor in your component, you need to call super().

24. What is the use of forceUpdate() and findDOMNode()?

forceUpdate(): It is used to updated the component manually.

findDOMNode():his method allows us to find or access the underlying DOM node.

25. Explain about React component lifecycle?

The lifecycle of the component is divided into four phases. They are:

- Initial Phase
- Mounting Phase
- Updating Phase
- Unmounting Phase

✓ **Initial Phase:** It is the birth phase of the lifecycle of a ReactJS component. In this phase, a component contains the default Props and initial State.

getDefaultProps(): It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.

getInitialState(): It is used to specify the default value of this.state. It is invoked before the creation of the component.

✓ **Mounting Phase:** In this phase, the instance of a component is created and inserted into the DOM.

componentWillMount(): This is invoked immediately before a component gets rendered into the DOM. In the case, when you call setState() inside this method, the component will not re-render.

componentDidMount(): This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.

render(): This method is defined in each and every component. It is responsible for returning a single root HTML node element. If you don't want to render anything, you can return a null or false value.

✓ **Updating Phase:** Here, we get new Props and change State. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again.

componentWillRecieveProps(): It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using this.setState() method.

shouldComponentUpdate(): It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.

render(): It is invoked to examine this.props and this.state and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number. If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.

componentDidUpdate(): It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

- ✓ **Unmounting Phase:** It is called when a component instance is destroyed and unmounted from the DOM.

  componentWillUnmount(): This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary cleanup related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

26. Difference between controlled and uncontrolled input forms?

Uncontrolled Component: The uncontrolled input is similar to the traditional HTML form inputs. The DOM itself handles the form data. Here, the HTML elements maintain their own state that will be updated when the input value changes. To write an uncontrolled component, you need to use a ref to get form values from the DOM. In other words, there is no need to write an event handler for every state update. You can use a ref to access the input field value of the form from the DOM.

```
import React, { Component } from "react";

export default class UnControlledForm extends Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);

    //defining the Ref
    this.input = React.createRef();
  }

  handleSubmit(event) {
    alert("A name was submitted: " + this.input.current.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" name="name" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Controlled Component: In HTML, form elements typically maintain their own state and update it according to the user input. In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with setState() method.

Controlled components have functions that govern the data passing into them on every onChange event, rather than grabbing the data only once, e.g., when you click a submit button. This data is then saved to state and updated with setState() method. This makes component have better control over the form elements and data.

```
export default class ControlledComp extends Component {
  constructor(props) {
    super(props);
    this.state = { value: "" };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({ value: event.target.value });
  }
  handleSubmit(event) {
    alert("You have submitted the input successfully: " + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <h1>Controlled Form Example</h1>
        <label>
          Name:
          <input
            type="text"
            value={this.state.value}
            onChange={this.handleChange}
          />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
```

27. **Explain different React event handlers you have used?**
    - ✓ onClick
    - ✓ onChange
    - ✓ onBlur
    - ✓ onFous
    - ✓ onSubmit
    - ✓ onMouseOut
    - ✓ onMouseOver

28. **Explain different ways of React Conditional Rendering?**

    - ✓ **If:** We can add if condition in functions and outside the return of render. Inside return we should use ternary operator

```
const isLoggedIn = props.isLoggedIn;
if (isLoggedIn) {
  return <UserLogin />;
}
return <GuestLogin />;
```

- ✓ **Logical && Operator**

```
{
   condition &&
   // whatever written after && will be a part of output.
}
```

- ✓ **Ternary Operator**

```
condition ?  true : false
```

- ✓ **Switch Case**

```
function NotificationMsg({ text}) {
  switch(text) {
    case 'Hi All':
      return <Message: text={text} />;
    case 'Hello JavaTpoint':
      return <Message text={text} />;
    default:
      return null;
  }
}
```

29. How to iterate the list data in React?

For iterating the array/list of objects data we will use the map method. While iterating we should mention the key to avoid the errors. Always give the unique value for the key attribute.

```
const myLists = ["Peter", "Sachin", "Kevin", "Dhoni", "Alisa"];

export default class ListIteration extends Component {
  render() {
    return (
      <div>
        {myLists.map((data) => (
          <h1 key={data}>{data}</h1>
        ))}
      </div>
    );
  }
}
```

30. Explain about React Ref?

Refs is the shorthand used for references in React. It is similar to keys in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

31. Explain about the fragments in ReactJS?

In React, whenever you want to render something on the screen, you need to use a render method inside the component. This render method can return single elements or multiple elements. The render method will only render a single root node inside it at a time. However, if you want to return multiple elements, the render method will require a 'div' tag and put the entire content or elements inside it. This extra node to the DOM sometimes results in the wrong formatting of your HTML output and also not loved by the many developers. **Fragments Will define fragments like <> </>**

```
class Columns extends React.Component {
 render() {
  return (
   <>
    <h2> Hello World! </h2>
    <p> Welcome to the JavaTpoint </p>
   </>
  );
 }
}
```

We will use the fragments instead of dive because of following reasons,

- ✓ It makes the execution of code faster as compared to the div tag.
- ✓ It takes less memory.

The shorthand syntax does not accept key attributes. You need a key for mapping a collection to an array of fragments such as to create a description list. If you need to provide keys, you have to declare the fragments with the explicit <React.Fragment> syntax.

```
Function  = (props) {
 return (
  <Fragment>
   {props.items.data.map(item => (
    // Without the 'key', React will give a key warning
    <React.Fragment key={item.id}>
     <h2>{item.name}</h2>
     <p>{item.url}</p>
     <p>{item.description}</p>
    </React.Fragment>
   ))}
  </Fragment>
```

32. **Explain about React Router?**

    Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

    React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications.

    As part of Router we will use the below library,

    react-router: It provides the core routing components and functions for the React Router applications.
    react-router-dom: It is used for web applications design.

33. **Explain about different types of Routers?**

    We have two types of Routers,

    - ✓ **BrowserRouter**: It is used to handle the dynamic URL.
    - ✓ **HashRouter**: It is used for handling static request

34. **Explain different components under Router?**

- ✓ **Route:** It is used to define and render component based on the specified path. It will accept components and render to define what should be rendered.

```
<Route path="/" component={App} />
```

- ✓ **Link:** This component is used to create links which allow to navigate on different URLs and render its content without reloading the webpage.

```
<li>
  <Link to="/about">About</Link>
</li>
```

- ✓ **NavLink:** It is same as link but it has extra feature like adding styles (**color**) to the link.

- ✓ **Switch:** This component is used to render components only when the path will be matched. Otherwise, it returns to the not found component.

- ✓ **Redirect:** This component is used to redirect to another route in our application

```
<Switch>
  <Route exact path="/" component={App} />
  <Route path="/about" component={About} />
  <Route path="/contact" component={Contact} />
  <Route component={Notfound} />
</Switch>
```

35. Give the steps to setup the router in application?

- ✓ Install the router dom package

```
$ npm install react-router-dom --save
```

- ✓ Add the Router (BrowserRouter) tag setup in **index.js** to make it available for all components

```
import { BrowserRouter as Router } from "react-router-dom";

ReactDOM.render(
  <Router>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Router>,
  document.getElementById("root")
);
```

✓ Create separate file for all routings and add that file in **App.js**

```
import React from "react";
import { Routes, Route } from "react-router-dom";
import ForgotPasswordStep1 from "../forgot-password/ForgotPasswordStep1";
import Login from "../login/Login";

export default function Routing() {

  return (
    <div>
      <Routes>
        <Route path="/" element={<Login />} />
        <Route path="/forgot" element={<ForgotPasswordStep1 />} />
      </Routes>
    </div>
  );
}
```

Including above file in App.js,

App.js × Routing.jsx

> JS App.js > App

```
1   import "bootstrap/dist/css/bootstrap.min.css";
2   import "react-datepicker/dist/react-datepicker.css";
3   import "./components/common/css/common.css";
4   import Header from "./components/common/Header";
5   import Footer from "./components/common/Footer";
6   import Routing from "./components/router/Routing";
7
8   function App() {
9
10    return (
11      <div>
12        <Header />
13        <Routing />
14        <Footer />
15      </div>
16    );
17  }
```

✓ Add the below logic in components to render the required one.

```
import { useNavigate } from "react-router-dom";

export default function Login() {

  //setting up navigate functionality
  let navigate = useNavigate();

  //Rendering success page. If user already exists then redirect him to open account page
  function renderSuccessPage(response) {
    if (response !== null) {
      if (localStorage.getItem("userType") === "existingUser") {
        navigate("/forgot");
      }
    }
  }
```

36. Which version of router you are using and give steps to route the pages?

Currently we are using Router 6. If we want to navigate from one component to other we need to import the useNavigate from react-router-dom package. We need to assign the useNavigate () to a variable and give the URL pattern in that. See the below example,

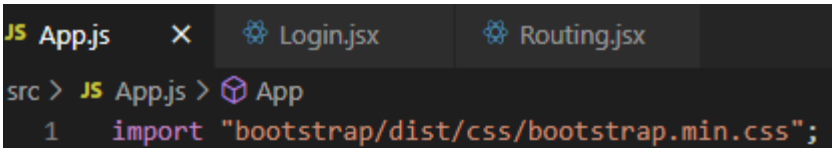
```
import { useNavigate } from "react-router-dom";

export default function Login() {

  //setting up navigate functionality
  let navigate = useNavigate();

  //Rendering success page. If user already exists then redirect him to open account page
  function renderSuccessPage(response) {
    if (response !== null) {
      if (localStorage.getItem("userType") === "existingUser") {
        navigate("/forgot");
      }
    }
  }
```

37. How to setup the BootStrap in ReactJS?

- ✓ Download the BootStrap libraries from NPM

```
npm install react-bootstrap bootstrap
```

- ✓ Add the **bootstrap.css in App.js** from installed library to make it available all over the components



- ✓ Import the required components from the installed library

```
import {
  Container,
  Row,
  Card,
  Form,
  FormGroup,
  FormControl,
  FormLabel,
  Button,
} from "react-bootstrap";
```

38. Difference between let, const and var?

- ✓ **Var:** We can define var in function scope only. Bet placement inside the functions.
- ✓ **Let:** We can define let in block and function scope. Best placement is loops.
- ✓ **Const:** Const is same like let but value can't be changed after assignment. Best place value never change.

39. Difference between npm install and npm install –-save?

There is no difference between these two. Before npm 5.0.0, it was necessary to add --save after package name because it will save the installed package to package.json file in the dependency section

It has a very frequently used command npm install [Package Name] –save. But the fact is there is no difference between npm install [Package Name] and npm install [Package Name] –save

40. Explain about Higher Order Component?

It is also known as HOC. In React, HOC is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component.

A higher order component function accepts another function as an argument. The map function is the best example to understand this. The main goal of this is to decompose the component logic into simpler and smaller functions that can be reused as you need.

HOC Rules:

- ✓ Do not use HOCs inside the render method of a component.
- ✓ HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument. If you add a ref to an element in the HOC component.

41. What is meant by code splitting in ReactJS?

Code-Splitting is a feature supported by bundlers like Webpack which can create multiple bundles that can be dynamically loaded at runtime.
As websites grow larger and go deeper into components, it becomes heavier. This is especially the case when libraries from third parties are included. Code Splitting is a method that helps to generate bundles that are able to run dynamically. It also helps to make the code efficient because the bundle contains all required imports and files.

Example: My project has two major modules sharing the common header and footer components. Header and footer having many files instead of loading these many files for each component we created it as library and started using the required components from that library.

42. How to build the own React Library?
    @TODO

43. Explain about React Context?

Context allows passing data through the component tree without passing props down manually at every level.

In React application, we passed data in a top-down approach via props. Sometimes it is inconvenient for certain types of props that are required by many components in the React application. Context provides a way to pass values between components without explicitly passing a prop through every level of the component tree.

There are two main steps to use the React context into the React application:

- ✓ Setup a context provider and define the data which you want to store.
- ✓ Use a context consumer whenever you need the data from the store

```
export default class ReactContectDemo extends Component {
  render() {
    /* Use a ContextProvider to pass the current theme, which allows every component to read it,
     no matter how deep it is. Here, we are passing the "dark" theme as the current value.*/
    return (
      <div>
        <ThemeContext.Provider value="dark">
          <Toolbar />
        </ThemeContext.Provider>
      </div>
    );
  }
}

// Now, it is not required to pass the theme down explicitly for every component.
function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

class ThemedButton extends React.Component {
  static contextType = ThemeContext;
  render() {
    return <Button theme={this.context} />;
  }
}
```

44. What is a React Hooks?

React Hooks is the new feature introduced in React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

45. When to use React Hooks?

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

46. Explain the rules of hooks?

Only call hooks at top level: Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions.

Only call hooks from React functions: You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components.

47. Explain different types of hooks you have used?

- ✓ useState
- ✓ useEffect
- ✓ useContext
- ✓ useNavigate
- ✓ useSelector
- ✓ useReducer
- ✓ useRef
- ✓ useMemo
- ✓ useCallBack
- ✓ useForm

48. Explain about useState?

Will use this hook for setting up and retrieving the data. It will store all types of data like objects, collections, and primitive types.

```jsx
import React, { useState } from "react";

export default function UseStateEx() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked on {count} times </p>
      <button onClick={() => setCount(count + 1)}>Counter</button>
    </div>
  );
}
```

49. Explain about useEffect hook?

useEffect used to performs the actions (side effects) in the functional components. It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

```jsx
import React, { useState, useEffect } from "react";
import { Container, Row, Col, Table } from "react-bootstrap";
import * as fectAPIData from "../redux/invokers/ServiceInvoker";
import serviceUrlData from "../shared/common/ServiceUrls.json";

export default function Student() {
  const [data, setData] = useState({});

//Invoking the API
  useEffect(() => {
    let serviceInput = {
      data: {
        studentId: profileData.data.id,
        belongsTo: profileData.data.belongsTo,
      },
      url: serviceUrlData.devURI + serviceUrlData.getMarksByIdURI,
      token: loggedUser.data.jwtToken,
      callbackSuccess: processSuccess,
      callbackFailure: processFailure,
    };

    //Invoking API
    fectAPIData.fetchAPIDataPost(serviceInput);
  }, []);
```

50. Explain about useNavigate hook?

If we want to navigate from one component to other we need to import the useNavigate from react-router-dom package. We need to assign the useNavigate () to a variable and give the URL pattern in that. See the below example,

```
import { useNavigate } from "react-router-dom";

export default function Login() {

  //setting up navigate functionality
  let navigate = useNavigate();

  //Rendering success page. If user already exists then redirect him to open account page
  function renderSuccessPage(response) {
    if (response !== null) {
      if (localStorage.getItem("userType") === "existingUser") {
        navigate("/forgot");
      }
    }
  }
```

51. Explain about useSelector hook?

We will use the useSelector hook to fetch the data from Redux. This hook is available from react-redux package.

```
import { useSelector } from "react-redux";

export default function Student() {
  //Pulling up the data from Redux
  const loggedUser = useSelector((state) => state.loginReducer.data);
  const profileData = useSelector((state) => state.profileReducer.data);
```

52. Explain about useContext hook?

useContext hook makes it easy to pass data throughout your app without manually passing props down the tree.

```jsx
import React, { useContext } from "react";

const ThemeContext = React.createContext("blue");

export default function UseContext() {
  return (
    <div>
      <ThemeContext.Provider value="dark">
        <Toolbar />
      </ThemeContext.Provider>
    </div>
  );
}

function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <Button theme={theme} />;
}
```

53. Explain about useRef hook?

In a React component, useState and useReducer can cause your component to re-render each time there is a call to the update functions. useRef() hook to keep track of variables without causing re-renders.

useRef doesn't notify you when its content changes.

```jsx
import React, { useRef, useEffect, useState } from "react";

export default function UseRefHook() {
  const initialData = {
    name: "",
  };
  const inputRef = useRef(null);
  const [data, setName] = useState(initialData);

  useEffect(() => {
    inputRef.current.focus();
  });

  const handleChange = () => {
    setName(inputRef.current.value);
  };

  return (
    <div>
      <input type="text" ref={inputRef} onChange={handleChange} />
      {data.name}
    </div>
  );
}
```

54. Explain about useMemo?

The useMemo is a hook used in the functional component of react that returns a memoized value. In Computer Science, memoization is a concept used in general when we don't need to recompute the function with a given argument for the next time as it returns the cached result. A memoized function remembers the results of output for a given set of inputs. For example, if there is a function to add two numbers, and we give the parameter as 1 and 2 for the first time the function will add these two numbers and return 3, but if the same inputs come again then we will return the cached value i.e 3 and not compute with the add function again. In react also, we use this concept, whenever in the React component, the state and props do not change the component and the component does not re-render, it shows the same output. The useMemo hook is used to improve performance in our React application.

55. Explain about useCallBack hook?

useCallBack is one of the new features introduced in the react hooks API. In simpler words, it returns a cached version of a function. Basically this hook is mainly use for performance reason (memory-wise).

```
const [height, setHeight] = useState(100)
const [age, setAge] = useState(3)

const handleSetHeight = () => setHeight(height + 10)
const handleSetAge = () => setAge(age + 1)
```

We setup two useState hooks & declare two functions to handle state changes. This seems normal. The issue here is that every time we invoke a function and re-render happens, a new instance of both of these functions will be created. Even if we invoke only one function, the instance of the other function will also be created. Imagine if there are more functions, how many instance have to be created during each re-render. It's redundant & causes performance issues.

useCallback helps in solving this issue. It will cache/memoized function that we pass to it. For example, let's rewrite both function above like this:

```
const handleSetHeight = useCallback(() => setHeight(height + 10), [height])
const handleSetAge = useCallback(() => setAge(age + 1), [age])
```

By doing this, whenever we invoke a function and re-render happens, a new function instance will only be created for that particular function that is being invoked. No new instance is created for the other function. The second argument passed to useCallback, the dependencies array plays a major part. A new function instance will only be generated if any value of the variable inside that array changes between re-rendering. If nothing changes, useCallback will just return the cached version of the function instance.

56. Difference between useMemo and useCallback?

useCallback gives you referential equality between renders for functions. And useMemo gives you referential equality between renders for values.

57. Explain about useReducer hook?

An alternative to useState. Accepts a reducer of type (state, action) => newState, and returns the current state paired with a dispatch method.

useReducer is usually preferable to useState when you have complex state logic that involves multiple sub-values or when the next state depends on the previous one. useReducer also lets you optimize performance for components that trigger deep updates because you can pass dispatch down instead of callbacks.

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

58. Explain about useForm Hook?

React Hook Form adopts the use of uncontrolled inputs using ref instead of depending on the state to control the inputs. This approach makes the forms more performance and reduces the number of re-renders.

Main motto of this to gather the inputs of from data and do the validations using yup schema.

For installing formhook we need to use below command

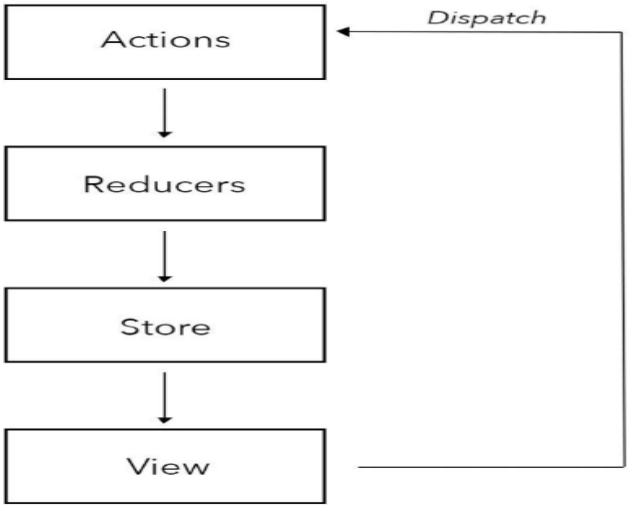
```
npm install react-hook-form
```

We will use the below format to read the data, validations with yup and displaying the error messages and form data submission,

```
import { useForm } from "react-hook-form";
import * as Yup from "yup";
import { yupResolver } from "@hookform/resolvers/yup";

export default function Login() {
  //Setting up the error messages.
  const validationSchema = Yup.object({
    customerId: Yup.string().required("Customer Id is Mandatory"),
    password: Yup.string().required("Password is Mandatory"),
  });

  //Validating the From with Yup
  const { register, handleSubmit, errors } = useForm({
    resolver: yupResolver(validationSchema),
  });

  //Submitting Form Data
  const onSubmit = (values) => {
    let loginRequest = {
      data: values
    };
  };
  return (
    <div>
      <Container>
        <Row className="justify-content-center mt-4">
          <Card style={{ width: "22rem" }}>
            <Card.Body>
              <Form onSubmit={handleSubmit(onSubmit)}>
                <FormGroup>
                  <FormLabel>Customer Id</FormLabel>
                  <FormControl
                    type="number"
                    placeholder="Enter your customer id"
                    name="customerId"
                    ref={register}
                    className={errors.customerId ? " is-invalid" : ""}
                  />
                  <div className="invalid-feedback">
                    {errors.customerId?.message}
                  </div>
                </FormGroup>
```

59. Explain about Redux?

Redux is a state management tool for JavaScript apps. We will use it to handle the global state for the react application. It is best suitable for handling client server applications.



Action: Action is static information about the event that initiates a state change. When you update your state with Redux, you always start with an action. Actions are in the form of Javascript objects, containing a type and an optional payload.

Reducer: A reducer is a pure function that takes care of inputting changes to its state by returning a new state. The reducer will take in the previous state and action as parameters and return the application state.

Redux store: The Redux store is the application state stored as objects. Whenever the store is updated, it will update the React components subscribed to it. You will have to create stores with Redux. The store has the responsibility of storing, reading, and updating state.

60. Explain about Redux setup?

Run the following command in your command prompt to install Redux.

```
npm install --save redux
```

To use Redux with react application, you need to install an additional dependency as follows –

```
npm install --save react-redux
```

To install developer tools for Redux, you need to install the following as dependency –

Run the below command in your command prompt to install Redux dev-tools.

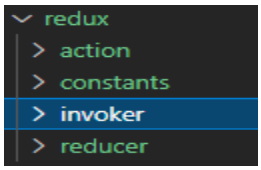
```
npm install --save-dev redux-devtools
```

If you do not want to install Redux dev tools and integrate it into your project, you can install **Redux DevTools Extension** for Chrome and Firefox.

- ✓ Set up the store information in the index.js file under src folder,

```
import { createStore, applyMiddleware } from "redux";
import { Provider } from "react-redux";
import thunkMiddleware from "redux-thunk";
import {composeWithDevTools} from 'redux-devtools-extension'

const store = createStore(rootReducer, composeWithDevTools(applyMiddleware(thunkMiddleware)));

ReactDOM.render(
    <React.StrictMode>
      <Provider store={store}>
        <App />
        </Provider>
      </React.StrictMode>,
  document.getElementById("root")
);
```

- ✓ Create the redux folder in components and add the separate folders for Actions, Reducers



- ✓ In the Actions files start adding the required functions of your own

components > redux > action > LoginAction.jsx > fecthLoggedInUserData

```
import {
  GET_LOGIN_REQUEST,
  GET_LOGIN_SUCCESS,
  GET_LOGIN_FAILURE,
} from "../constants/InternetConstants";

export function fetchLoginRequest() {
  return {
    type: GET_LOGIN_REQUEST,
  };
}

export function fecthLoggedInUserData(userData) {
  return {
    type: GET_LOGIN_SUCCESS,
    payload: userData,
  };
}

export function fetchUserError(error) {
  return {
    type: GET_LOGIN_FAILURE,
    payload: error,
  };
}
```

- ✓ In the reducer function invoke the redux store by using the actions from corresponding Action file and return the new data in the state from redux store.

```
import {
  GET_LOGIN_REQUEST,
  GET_LOGIN_SUCCESS,
  GET_LOGIN_FAILURE,
} from "../constants/InternetConstants";

const initialData = {
  isLoading: false,
  data: {},
  error: "",
};

export const loginReducer = (state = initialData, action) => {
  switch (action.type) {
    case GET_LOGIN_REQUEST:
      return {
        ...state,
        isLoading: true,
      };
    case GET_LOGIN_SUCCESS:
      return {
        isLoading: false,
        data: action.payload,
        error: "",
      };
    case GET_LOGIN_FAILURE:
      return {
        isLoading: false,
        data: {},
        error: action.payload,
      };
    default:
      return state;
  }
};

export default loginReducer;
```

- ✓ If you have more than one reducer combine them in one file and provide the reducer file as input to the store in store setup.

index.js M index.jsx U

> components > redux > reducer > index.jsx > ...

```
1   import { combineReducers } from "redux";
2   import LoginReducer from "./LoginReducer";
3
4   const allReducers = combineReducers({
5     loginReducer: LoginReducer,
6   });
7
8   export default allReducers;
```

✓ Passing about root reduce (All reducers combined) file to the store as input.

```
import { createStore, applyMiddleware } from "redux";
import { Provider } from "react-redux";
import thunkMiddleware from "redux-thunk";
import {composeWithDevTools} from 'redux-devtools-extension'
import rootReducer from "./components/redux/reducer";

const store = createStore(rootReducer, composeWithDevTools(applyMiddleware(thunkMiddleware)));
```

✓ Now trigger the request from UI component to store,

components > login > Login.jsx > Login

```
import { useDispatch } from "react-redux";

export default function Login() {

  //Setting up useDispatch for Redux hit
  let dispatch = useDispatch();

  //Submitting Form Data
  const onSubmit = (values) => {
    setErrorMessage(null);
    let loginRequest = {
      url: serviceUrlData.devURI + serviceUrlData.loginURI,
      data: values,
      callbackSuccess: renderLoggedCustomer,
      callbackFailure: renderTheErrorPage,
    };
    //Invoking the API
    dispatch(authenticateUser(loginRequest));
  };
```

✓ When we dispatch the action request will start from action component

```
import {
    fetchLoginRequest,
    fecthLoggedInUserData,
    fetchUserError,
  } from "../action/LoginAction";
import axios from "axios";

const headers = {
    "Content-Type": "application/json",
  };

  const authenticateUser = (input) => {
    return function (dispatch) {
      dispatch(fetchLoginRequest());
      axios
        .post(input.url, JSON.stringify(input.data), { headers: headers })
        .then((response) => {
          dispatch(fecthLoggedInUserData(response));
          input.callbackSuccess(response.data);
        })
        .catch((error) => {
          dispatch(fetchUserError(error.message));
          input.callbackFailure(error)
        });
    };
  };

  export default authenticateUser;
```

61. Difference between Redux and Redux Thunk?

Redux itself is synchronous if you want to perform the async calls we will use the redux thunk.

62. Explain about Redux Thunk setup?

Redux Thunk can be installed by running

npm install redux-thunk

```
import { createStore, applyMiddleware } from "redux";
import { Provider } from "react-redux";
import thunkMiddleware from "redux-thunk";
import {composeWithDevTools} from 'redux-devtools-extension'

const store = createStore(rootReducer, composeWithDevTools(applyMiddleware(thunkMiddleware)));

ReactDOM.render(
  <Router>
    <React.StrictMode>
    <Provider store={store}>
      <App />
      </Provider>
    </React.StrictMode>
  </Router>,
  document.getElementById("root")
);
```

63. How to perform the Async calls with Redux?

We were doing the Async calls with the help of **Promise** api.

```
//Get the Internal Accounts
function getInternalAccounts(input,headers) {
  return axios.post(internalAccountsUrl, JSON.stringify(input.data), {
    headers,
  });
}

//Get the External Accounts
function getExternalAccounts(input,headers) {
  return axios.post(externalAccountsUrl, JSON.stringify(input.data), {
    headers,
  });
}


//Invoking Async Operations
export function getCustomerAccounts(input) {
    const headers = {
        Authorization: "Bearer " + input.token,
        "Content-Type": "application/json",
      };

  Promise.all([
    getInternalAccounts(input,headers),
    getExternalAccounts(input,headers),
  ])
    .then(function (results) {
      input.callbackSuccess(results);
    })
    .catch(function (error) {
      console.log(error);
      input.callbackFailure(error);
    });
}
```

- ✓ We need to pull the data from Async response like below,

```
//Rendering success page
function renderSuccessPage(results) {
  if (results !== null) {
    let [internalAccounts, externalAccounts] = results;
    setInternalAccounts(internalAccounts.data);
    setExternalAccounts(externalAccounts.data);
  }
}
```

64. Explain about axios?

Axios is a library that helps us make http requests to external resources. Axios uses methods like get() and post() that perform http GET and POST requests for retrieving or creating resources.

With the help of this we can call the external service and we can process the response of service and also the exceptions of external services.

```
export function fetchAPIDataPost(input) {
  const headers = {
    Authorization: "Bearer " + input.token,
    "Content-Type": "application/json",
  };

  axios
    .post(input.url, JSON.stringify(input.data), {
      headers,
    })
    .then((response) => {
      console.log("Final Response: ", response.data);
      input.callbackSuccess(response.data);
    })
    .catch((error) => {
      console.log("Error Message is: ", error.message);
      input.callbackFailure(error);
    });
}
```

65. Difference between axios and fetch?

- ✓ Fetch and Axios are very similar in functionality, but for more backwards compatibility Axios seems to work better (fetch doesn't work in IE 11).
- ✓ Axios allows the timeouts and Async calls but Fetch does not support timeout and Async calls.
- ✓ Axios parses the JSON response automatically but Fetch we need to use the .json() method to get parsed.
- ✓ We need to install axios but Fetch will come by default.
- ✓ Axios contains data as object format but Fetch will return response as stringified.

66. Explain about Pure Components?

Now, ReactJS has provided us a Pure Component. If we extend a class with Pure Component, there is no need for shouldComponentUpdate() Lifecycle Method. ReactJS Pure Component Class compares current state and props with new props and states to decide whether the React component should re-render itself or Not.

In simple words, if the previous value of state or props and the new value of state or props is the same, the component will not re-render itself. Since Pure Components restricts the re-rendering when there is no use of re-rendering of the component. Pure Components are Class Components which extends React.PureComponent.

```
import React from 'react';

export default class Test extends React.PureComponent{
   render(){
      return <h1>Welcome to GeeksforGeeks</h1>;
   }
}
```

67. Difference between Context and Redux?

Context API prompts a re-render on each update of the state and re-renders all components regardless. Redux however, only re-renders the updated components.

Context API: Resourceful and ideal for small applications where state changes are minimal
Redux: Perfect for larger applications where there are high-frequency state updates